BABEŞ-BOLYAI TUDOMÁNYEGYETEM
MATEMATIKA ÉS INFORMATIKA KAR

Licenszvizsga
Informatika - magyar nyelv

1. VÁLTOZAT

MEGJEGYZÉSEK

- Mindegyik tétel kötelező; a tételeknél teljes megoldásokat kérünk.
- A dolgozat minimális átmenő jegye az ötös (5.00).
- Munkaidő: három (3) óra.

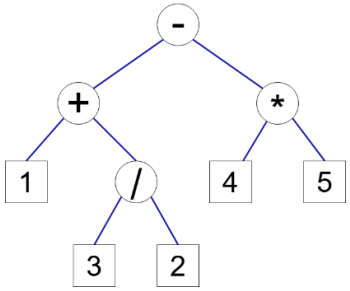
Algoritmusok és programozás TÉTEL

- **A használt programozási nyelvet meg kell jelölni.**
- **A megírandó programokban nem szükséges a dinamikusan lefoglalt memória felszabadítása.**
- **A megfelelő programozási stílus hiánya (azonosítók beszédes elnevezése, indentálás, megjegyzések ha szükségesek, a kód olvashatósága) az adott tételhez tartozó pontszám 10%-nak az elvesztéséhez vezet.**
- **Tilos új adattagok és metódusok hozzáadása a kijelentésben említetteken kívül, leszámítva a konstruktorokat. Tilos a kijelentésben megadott adattagok és metódusok láthatóságának a megváltoztatása.**
- **Előre definiált rendezett konténerek és rendezési valamint keresési műveletek használata tilos.**
- **Adattípusokhoz használhatóak a megfelelő programozási nyelvek (Python, C++, Java, C#) létező könyvtárai.**

Egy számtani kifejezés ábrázolható egy bináris fa formájában a következőképpen. Ha a kifejezés egyetlen operandusból áll, akkor a megfelelő bináris fának egyetlen csúcsa van (a gyökér), amely által tárolt hasznos információ az adott operandus. Ha a kifejezés **E1** *op* **E2** formájú, ahol *op* egy bináris operátor és **E1** és **E2** számtani kifejezések, akkor a megfelelő bináris fa gyökere az *op* operátort tárolja hasznos információként, a bal részfa az **E1** kifejezésnek felel meg és a jobb részfa az **E2** kifejezésnek felel meg.

Egy kifejezésnek megfelelő bináris fa *posztorder* bejárása megadja annak *posztfix alakját*. ***Például***, a (1+3/2)-4*5 kifejezés esetén

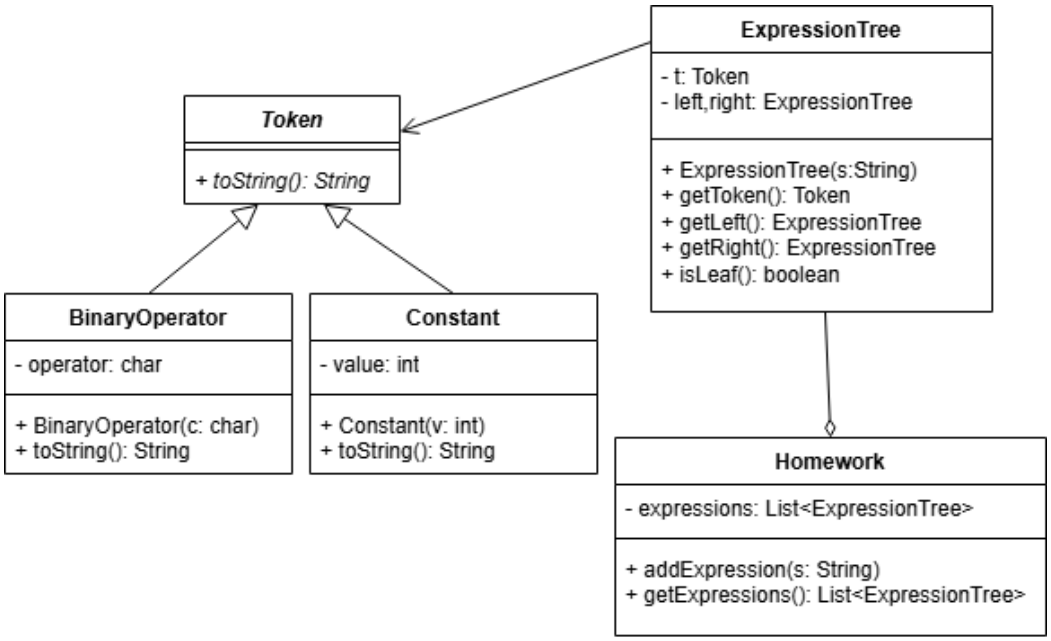
- a megfelelő bináris fa



- a *posztfix alak* '132/+45*-'.

1. **(3.5 pont)** Írjatok függvényt a **Python, C++, Java** vagy **C#** programozási nyelvek valamelyikében, amely paraméterként megkapja a legtöbb 100 karakterből álló *s* sztringet, amely egy számtani kifejezés **helyes** *posztfix alakját* jelöli, mely a '+' (összeadás), '-' (kivonás), '*' (szorzás), '/' (valós osztás) operátorokat tartalmazhatja és operandusként egy számjegyből álló nullától különböző számokat. A függvény vissza kell térítse a kifejezés értékét. Az algoritmus időbonyolultsága $\theta(n)$ kell legyen, ahol n az *s* karakterlánc hossza. Azok a megoldások, amelyek nem tartják be a kért időbonyolultságot, részpontszámot érnek. **Példa:** a '132/+45*-' sorozat esetén a visszatérített érték **-17.5**. Segédfüggvények implementálása megengedett.
2. **(1 pont)** Adjátok meg az **1**-es pontnál megírt függvény időbonyolultságát a *legjobb*, *átlag* és *legrosszabb* esetben. Indokoljátok a választ.

3. **(4.5 pont)** Adott az alábbi UML diagram, mely tartalmazza az **ExpressionTree** (egy helyes aritmetikai kifejezéshez tartozó bináris fa), ***Token*** (Szimbólum), **BinaryOperator** (BinárisOperátor), **Constant** (Konstans) és **Homework** (HáziFeladat) osztályokat.



- Az **ExpressionTree** osztály egy számtani kifejezésnek megfelelő bináris fát ábrázol, amelynek operandusai konstansok (nem nulla természetes számok), és operátorai binárisak (+, -, *, /).
  - Az **ExpressionTree** osztály konstruktora egy nem üres *s* karakterláncot kap paraméterként, amely egy helyes számtani kifejezés: zárójeleket, bináris operátorokat és konstans operandusokat tartalmaz. A konstruktor felépíti az *s* kifejezésnek megfelelő bináris fát. *Ezt a konstruktort nem kell implementálni*.
  - Az **isLeaf()** metódus igazat térít vissza, ha a bináris fának egyetlen csúcsa van (azaz egyetlen operandust tartalmazó számtani kifejezést ábrázol), és hamisat egyébként.
  - A **getToken()** metódus visszatéríti a számtani kifejezést ábrázoló **A** bináris fa gyökerében tárolt hasznos információt, míg a **getLeft()** és **getRight()** metódusok visszatérítik az **A** fa bal, illetve jobb részfáját.
- A **Token** absztrakt osztály a bináris fa csúcsaiban tárolt hasznos információt ábrázolja (bináris operátor vagy konstans operandus). Az osztálynak van egy **toString()** absztrakt metódusa, amely visszatéríti a szimbólumhoz tartozó karakterláncot: ha a szimbólum egy bináris operátor, akkor az operátort ábrázoló karaktert téríti vissza (pl. '+'); ha a szimbólum egy konstans, akkor a konstans értékét karakterláncként (pl. '123').
- A **Homework** osztály egy **expressions** nevű listában tárolja el azokat a számtani kifejezéseket, amelyeket egy diák házi feladatként kapott. A **Homework** osztály **addExpression(s: String)** metódusa egy *s* karakterláncot kap paraméterként, amely egy helyes számtani kifejezést ábrázol, és hozzáadja a *s*-nek megfelelő bináris fát az **expressions** lista végéhez. A **getExpressions()** metódus visszatéríti a számtani kifejezéseknek megfelelő bináris fák listáját.

Írjatok programot a **C++, Java** vagy **C#** programozási nyelvek valamelyikében, az alábbi követelmények szerint:

a) Deklaráljátok az összes osztályt, adattagot és metódust a fenti diagramnak megfelelően. Csak a következő metódusokat kell implementálni:

> **a1)** A **BinaryOperator** osztály konstruktorát. Az **operator** adattag egy számtani bináris operátort kell hogy ábrázoljon (+, -, *, /). A konstruktor érvényesíti ezt a feltételt.
> **a2)** Az **ExpressionTree** osztály **isLeaf()** metódusát.
> **a3)** A **Homework** osztály **addExpression(s*:* String)** metódusát.

b) Definiáljatok egy függvényt, amely paraméterként egy **ExpressionTree** objektumot kap, és visszatérít egy karakterláncot, amely a fa által tárolt számtani kifejezés posztfix alakját ábrázolja.

c) Hozzatok létre egy **Homework** típusú *h* objektumot, amelyhez adjátok hozzá a következő számtani kifejezéseket kiértékelés céljából: '(5-3)+(2*6)', '5-(3*4+5)/(6-2)' és '3*(5/(3+2)-4)+6'. Írjátok ki mintden *h*-ban tárolt kifejezés posztfix alakját, a **b)** pontban definiált függvény segítségével.

Adatbázisok TÉTEL **1. VÁLTOZAT**

1. Feladat (4 pont)

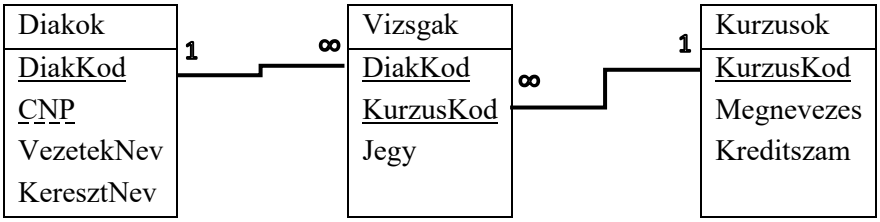
Egy törölközőket gyártó cég egy relációs adatbázisban szeretné tárolni a klienseire-, a cég által gyártott törölközőkre vonatkozó információkat, valamint a kliensek törölközőkre adott értékeléseit:

- Kliensek esetén tároljuk: kliens kódját, nevét, e-mail címét (egyedi minden kliens esetén), születési dátumát.
- Egy törölköző esetén érdekel: törölköző kódja, megnevezése, leírása, modellje, hosszúsága és szélessége.
- Egy alapanyagnak van: kódja, megnevezése és leírása. Egy alapanyag felhasználható több törölköző gyártásához is, míg egy törölköző többféle alapanyagból is készülhet. Minden alapanyag legfennebb egyszer társítható egy adott törölközőhöz. Minden alapanyag-törölköző páros esetén (pl. a t1 törölköző és az a1 alapanyag párosa) tárolni kell azt is, hogy az adott alapanyag milyen (százalékos) arányban van jelen a törölköző összetételében.
- Egy kliens több törölközőt is értékelhet, míg egy törölköző is több klienstől kaphat értékelést. Minden kliens legfennebb egy értékelést adhat egy adott törölközőre.

Tervezzük meg a fenti információknak megfelelő relációs adatbázis sémáját, melynek táblái BCNF-ben vannak. Jelöljük az elsődleges- és külső kulcsokat, valamint a relációk további kulcsjelöltjeit. A relációk sémáját az alábbiakban megadott módok egyikének megfelelően adjuk meg:

* példa a Diakok, Kurzusok és Vizsgak táblákra vonatkozóan:

V1. Adatbázis-diagram a táblák megadásával: az elsődleges kulcsok egyenes vonallal, míg a tábla további kulcsjelöltjei szaggatott vonallal vannak aláhúzva; a fiú táblában megadott külső kulcsot az apa táblában szereplő elsődleges kulccsal/kulcsjelölttel egy direkt él köti össze (lsd. a Vizsgak tábla DiakKod attribútuma és a Diakok tábla DiakKod attribútuma közötti él).



V2.

Diakok[DiakKod, CNP, Vezeteknev, Keresztnev]

Kurzusok[KurzusKod, Megnevezes, Kreditszam]

Vizsgak[DiakKod, KurzusKod, Jegy]

Az elsődleges kulcsok alá vannak húzva egyenes vonallal, míg a további kulcsjelöltek szaggatott vonallal. {DiakKod} külső kulcs a Vizsgak relációban, mellyel a Diakok tábla {DiakKod} attribútumára hivatkozunk. {KurzusKod} külső kulcs a Vizsgak relációban, mellyel a Kurzusok tábla {KurzusKod} attribútumára hivatkozunk.

2. Feladat (5 pont)

Tekintsük az alábbi relációsémákat, melyek egy társasjátékok kölcsönzésével foglalkozó cégre vonatkozó információkat tárolnak:

Kategoriak(KategoriaKod, Megnevezes)

Tarsasjatekok(TarsasjatekKod, Megnevezes, *KategoriaKod*, MegjelenesiEv, MinJatekosSzam, MaxJatekosSzam, NapiAr)

Szemelyek(SzemelyKod, Nev, Telefonszam)

Berlesek(BerlesKod, *TarsasjatekKod*, *SzemelyKod*, BerlesDatuma, BereltNapokSzama, Vegosszeg)

Az elsődleges kulcsok alá vannak húzva, míg a külső kulcsok dőlt betűvel vannak szedve, nevük pedig megegyezik a hivatkozott tábla elsődleges kulcsának nevével.

a. Írjunk egy SQL lekérdezést, mely megadja azon társasjátékok kódját és megnevezését, amelyeket egyedül is lehet játszani (lsd. MinJatekosSzam attribútum), ki voltak bérelve legalább egyszer a 2024-es évben, valamint amelyek megjelenési éve (lsd. MegjelenesiEv attribútumot) megegyezik az előző feltételeknek eleget tevő, legfrissebben (legkésőbb) kiadott társasjátékok megjelenési évével!

b. Tekintsük a Tarsasjatekok, Kategoriak, Szemelyek és Berlesek relációk alábbi előfordulásait:

Szemelyek:

SzemelyKod	Nev	Telefonszam
1	Sz1	1111111111
2	Sz2	2222222222
3	Sz3	3333333333

Kategoriak:

KategoriaKod	Megnevezes
1	stratégiai
2	családi
3	kooperatív

Tarsasjatekok:

TarsasjatekKod	Megnevezes	KategoriaKod	MegjelenesiEv	MinJatekosSzam	MaxJatekosSzam	NapiAr
1	TJ1	1	2020	3	6	15
2	TJ2	2	2016	2	4	5
3	TJ3	1	2020	2	5	12
4	TJ4	3	2024	3	10	10

Berlesek:

BerlesKod	TarsasjatekKod	SzemelyKod	BerlesDatuma	BereltNapokSzama	Vegosszeg
1	1	1	2025.05.15	3	45
2	2	1	2025.05.18	5	25
3	3	2	2025.05.10	2	24
4	3	2	2025.06.02	3	36
5	1	3	2025.05.25	1	15
6	2	3	2025.06.03	4	20
7	3	3	2025.06.10	2	24
8	4	2	2025.06.20	3	30

b1. A fentebb megadott reláció előfordulások esetén adjuk meg az alábbi lekérdezés kiértékelésének eredményét. Adjuk meg az oszlop(ok) nevét és értékét! A választ NEM kell megindokolni!

```
SELECT Sz.SzemelyKod, Sz.Nev, MIN(Vegosszeg) MinOsszeg
FROM Szemelyek Sz
    INNER JOIN Berlesek B ON Sz.SzemelyKod = B.SzemelyKod
    INNER JOIN Tarsasjatekok TJ ON B.TarsasjatekKod = TJ.TarsasjatekKod
    INNER JOIN Kategoriak K ON TJ.KategoriaKod = K.KategoriaKod
WHERE K.Megnevezes = 'stratégiai'
GROUP BY Sz.SzemelyKod, Sz.Nev
HAVING COUNT(DISTINCT B.TarsasjatekKod) =
    (SELECT COUNT(*)
     FROM Tarsasjatekok TJ2
          INNER JOIN Kategoriak K2 ON TJ2.KategoriaKod = K2.KategoriaKod
     WHERE K2.Megnevezes = 'stratégiai'
     )
```

b2. Döntsük el, hogy a Berlesek reláció fentebb megadott előfordulása kielégíti-e az alábbi funkcionális függőségeket vagy sem! Indokoljuk meg a választ!

- {BerlesKod} → {TarsasjatekKod}
- {SzemelyKod} → {BerlesDatuma}.

1. VÁLTOZAT

Operációs rendszerek TÉTEL

1. Feladat (5 pont) Válaszoljon az alábbi program végrehajtásával kapcsolatos kérdésekre, feltéve, hogy minden szükséges fejlécállomány csatolva van.

```
 1  int main() {
 2      int k;
 3      if(fork() < 0) {
 4          printf("abc\n");
 5      } else if(fork() == 0) {
 6          return 1;
 7      } else {
 8          return 2;
 9      }
10      while((k = wait(NULL)) > 0) {
11          printf("%d\n", k);
12      }
13      return 3;
14  }
```

a) Magyarázzuk el részletesen a 3. sor működését.

b) Magyarázzuk el részletesen a 10-12 sorok működését.

c) Mi lesz a standard kimenetre írva, ha mindkét `fork` hívás hibával tér vissza? Indokoljuk a választ.

d) Mi lesz a standard kimenetre írva, ha mindkét `fork` hívás sikeresen hajtódik végre? Indokoljuk a választ.

e) Mit ír a standard kimenetre az eredeti folyamat, amennyiben a 8. sort töröljük, és mindkét `fork` hívás sikerrel hajtódik végre? Indokoljuk a választ.

2. Feladat (4 pont) Válaszoljon az alábbi `./a.sh` szkript végrehajtásával kapcsolatos kérdésekre.

```
 1  #!/bin/bash
 2
 3  N=$1
 4  D=$2
 5  shift 2
 6
 7  for A in $*; do
 8      if [ $N -gt 0 ]; then
 9          mkdir -p $D/$A
10          $0 `expr $N - 1` $D/$A $*
11      else
12          echo abc > $D/$A
13      fi
14  done
```

a) Magyarázzuk el részletesen a 10. sor működését

b) Milyen értékeket vesz fel az `A` változó az alábbi parancs végrehajtásakor?

```
./a.sh 1 a b c d
```

c) Feltéve, hogy az `x` katalógus nem létezett eredetileg, mi lesz a standard kimenetre írva az alábbi parancsok végrehajtását követően?

```
./a.sh 1 x a b
find x -type f | sort
```

d) Feltéve, hogy az `y` katalógus nem létezett eredetileg, mi lesz a standard kimenetre írva az alábbi parancsok végrehajtását követően?

```
./a.sh 2 y a b
cat `find y -type f` | wc -l
```

BAREM INFORMATICĂ
VARIANTA 1

Subiect Algoritmică și Programare

Oficiu – **1p**

Cerința **1**. **– 3.5p**
- signatura – **0.2p**
- implementare având complexitate timp $\theta(n)$ – **3.2p**
  * soluție având complexitate timp $O(n^2)$ – **2**p
- returnare rezultat – **0.1p**

Cerința **2**. **– 1p**
- caz favorabil **0.4p** din care
  complexitate – 0.2
  justificare – 0.2
- caz mediu **0.4p** din care
  complexitate – 0.2
  justificare – 0.2
- caz defavorabil **0.2p** din care
  complexitate – 0.1
  justificare – 0.1

Cerința **3.a) – 2.25p**

Definirea clasei Token – **0.2p** din care
- clasă abstractă – 0.1
- metoda **toString** – 0.1

Definirea clasei BinaryOperator– **0.65p** din care
- relația de moștenire – 0.15
- atribut – 0.1
- **constructor (a1)** – 0.3
- metoda **toString** – 0.1

Definirea clasei Constant– **0.45p** din care
- relația de moștenire – 0.15
- atribut – 0.1
- constructor – 0.1
- metoda **toString** – 0.1

Definirea clasei ExpressionTree – **0.55p** din care
- atribute– 0.1
- constructor – 0.1
- metode **getToken, getLeft, getRight** – 0.15
- metoda **isLeaf (a2) –** 0.2

Definirea clasei Homework – **0.4p** din care
- atribut – 0.1
- metoda **getExpressions** – 0.1p
- metoda **addExpression (a3)** – 0.2p

Funcția **3.b)** – **1.75p**
- signatura – **0.1p**
- implementare parcurgere postordine – **1.55p**
- returnare rezultat – **0.1p**

Funcția principală **3.c)** – **0.5p**
- construire obiect **h** – 0.05p
- adăugare expresii aritmetice în **h** - 0.25p
- parcurgere listă expresii memorate în **h** și afișare formă postfixată – 0.2p din care
  apel funcție b) pentru expresie– 0.1p
  afișare formă postfixată expresie - 0.1 p

BAREM INFORMATICĂ
VARIANTA 1

Subiect Baze de date

Oficiu – **1p**

Problema 1. Punctaj - 4p

- relații cu atribute corecte, chei primare, chei candidat: **3p**
- legături modelate corect (chei externe): **1p**

Problema 2. Punctaj - 5p

- **a** - rezolvarea completă a interogării: **2.5p**

- **b1** - rezultat evaluare interogare:

CodPersoana	Nume	MinSum
3	P3	15

  - coloane – **0.5p**
  - valori tuplu – **1p**

- **b2 -** {CodInchiriere} → {CodJocDeSocietate} este satisfăcută – **0.25p**; **0.25p** explicație
  - {CodPersoana} → {DataInchirierii} nu este satisfăcută – **0.25p**; **0.25p** explicație

Notă: La specializările Informatică engleză și Informatică maghiară se iau în considerare versiunile traduse în limbile corespunzătoare.

VARIANTA 1

SUBIECT Sisteme de operare

Barem:

1p – oficiu

Problema 1 (5p)

1p – a) Se rulează `fork` şi dacă eşuează condiţia va fi adevărată. Dacă `fork` nu eşuează, se creează un proces fiu şi condiţia va fi evaluată ca falsă şi de procesul părinte şi de procesul fiu.

1p – b) Câtă vreme `wait` returnează succes, se tipăreşte PID-ul procesului fiu care tocmai s-a încheiat.

1p – c) Se va afişa `abc` pentru că procesul iniţial va intra în primul if, dar nu va intra în `while` pentru că neavând procese fiu, `wait` va returna eroare.

1p – d) Nu se va afişa nimic, pentru că procesul iniţial şi primul proces fiu se încheie la linia 8, al doilea proces fiu şi procesul nepot se incheie la linia 6

1p – e) Afişează PID-urile celor două procese fiu pe care le creează la liniile 3 şi 5.

Problema 2 (4p)

1p – a) Scriptul Shell se relansează în execuţie, cu primul argument decrementat, al doilea argument directorul tocmai creat şi restul argumentelor identice cu cele rămase după shift.

1p – b) Va lua valorile `b`, `c` şi `d`

1p – c) Se va afişa: `x/a/a  x/a/b  x/b/a  x/b/b`

1p – d) Se va afişa cifra 8